На предыдущем занятии мы с вами познакомились с новым видом ссылок на rvalue выражения, которые записываются по синтаксису:

```
<тип данных>&& <имя r-ссылки> = <rvalue значение>;
```

Такие ссылки появились в стандарте C++11 и, спрашивается, для чего они нужны и какую роль играют при написании программ? В этом занятии мы с вами, как раз, ответим на этот вопрос.

Начнем с простого примера. Допустим, нам нужно прописать класс DArray для работы с динамическим массивом. Мы с вами это уже делали на одном из прошлых занятий, здесь, я запишу его в очень упрощенном виде следующим образом:

```cpp
class DArray {
    int *data {nullptr};
    int length {0};
    int capacity {0};

public:
    DArray(int size = 0) : length(0), capacity(size)
    {
        std::cout << "DArray create" << std::endl;
        data = new int[capacity];
    }

    DArray(const DArray& other) : length(other.length), capacity(other.capacity)
    {
        std::cout << "DArray copy" << std::endl;
        data = new int[capacity];
        for(int i = 0; i < length; ++i)
            data[i] = other.data[i];
    }

    ~DArray()
    {
        std::cout << "DArray delete" << std::endl;
        delete[] data;
    }

    const DArray& operator=(const DArray& right)
    {
        if(this == &right)
```

```
            return *this;

        std::cout << "DArray assigment" << std::endl;

        length = right.length;
        capacity = right.capacity;

        delete[] data;
        data = new int[capacity];

        for(int i = 0; i < length; ++i)
            data[i] = right.data[i];

        return *this;
    }
};
```

Здесь присутствуют три поля: length – число записанных данных; capacity – физический размер массива; data – указатель на область памяти с данными. Затем, идет конструктор по умолчанию, конструктор копирования и деструктор. В конце прописано переопределение операции присваивания. Все методы выводят в консоль соответствующие сообщения (строки).

Давайте теперь представим, что объявляется функция для формирования объекта класса DArray с заданной длиной массива:

```language-cpp
DArray create_array(int size)
{
    DArray ar(size);
    return ar;
}

А в функции main создается массив с помощью функции create_array:

int main()
{
    DArray ard;
    ard = create_array(10);

    return 0;
}
```

Если сейчас запустить эту программу (в Visual Studio 2019), то в консоли увидим следующие строчки:

DArray create
DArray create
DArray copy
DArray delete
DArray assigment
DArray delete
DArray delete

Объект класса DArray был создан три раза: первый раз в функции main, второй раз – в функции create_array и третий раз – при передаче объекта ar из функции create_array (сработал конструктор копирования). Кроме того, была выполнена переопределенная операция присваивания, где данные массива также копируются. И того четыре ресурсоемкие операции по созданию и копированию данных объекта класса DArray.

Конструктор перемещения

Спрашивается, можно ли как-то улучшить этот код и повысить его производительность? Очевидно, да. И первое, что напрашивается – это пробросить временный объект ar из функции create_array сразу в функцию main без его копирования. Но как это сделать? Просто записать ссылку не получится:

```language-cpp
DArray& create_array(int size) ...
```

так как она в итоге будет вести на удаленный объект и программа завершится аварийно. Поэтому мы оставим формально операцию копирования:

```language-cpp
DArray create_array(int size) ...
```

но в класс DArray добавим еще один конструктор специального вида – **конструктор перемещения**:

```language-cpp
class DArray {
...
    DArray(DArray&& move) noexcept : length(move.length),
capacity(move.capacity)
    {
        std::cout << "DArray move" << std::endl;
        data = move.data;
        move.data = nullptr;
```

```
    }
...
};
```

Чаще всего он записывается именно так. В качестве параметра rvalue-ссылка на объект класса DArray, а сам конструктор помечается ключевым словом noexcept, чтобы компилятор мог его совершенно свободно использовать для любых подходящих целей.

Что это за конструктор и какова его роль? **С его помощью также создается новый объект класса DArray, но при его вызове предполагается, что все данные объекта move будут просто «забираться» новым созданным объектом, без копирования**. В частности, именно так происходит с массивом data. Мы лишь сохраняем указатель на ту же область памяти, что и в объекте move, не создавая ее копию. Тем самым, сокращаем объем вычислений. В конце указатель data для объекта move устанавливается в значение nullptr, чтобы при удалении этих двух объектов (move и нового созданного) память data не освобождалась дважды. В этом и состоит суть перемещения объекта с помощью конструктора перемещения. Что именно и как будет перемещаться, решает сам программист, при реализации этого конструктора. Но в любом случае создается новый объект класса (в нашем случае DArray). Прежний объект move считается более не используемым и при проектировании программы это должно соблюдаться.

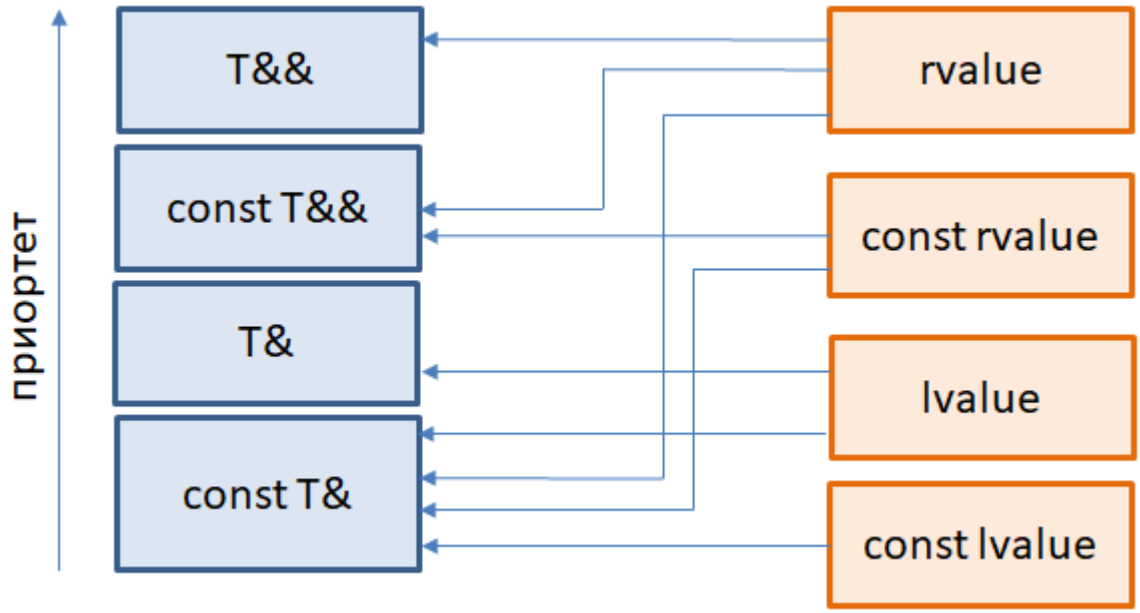
Теперь, при запуске программы, в консоли появляются строчки:

```
DArray create
DArray create
DArray move
DArray delete
DArray assigment
DArray delete
DArray delete
```

Объект был создан дважды, при третьем создании использовался конструктор перемещения и объем вычислений здесь, как правило, существенно ниже, чем при копировании объекта. Причем, компилятор сам выбрал именно конструктор перемещения, а не копирования. Это связано с тем, что возвращаемое значение оператора return воспринимается как rvalue выражение. И ему, как раз соответствует конструктор перемещения.

Если же возникает возможность выбора одного конструктора из множества, то компилятор руководствуется следующими простыми правилами

приоритетов использования ссылок:



Отсюда хорошо видно, что если можно использовать или конструктор копирования или конструктор перемещения, то будет выбран конструктор перемещения, как наиболее приоритетный. Учитывая, что возвращаемое из функции значение воспринимается как rvalue, то у компилятора появляется возможность выбора и он выбирает конструктор перемещения.

Правило пяти

Также обратите внимание, что при объявлении конструктора перемещения, стандартные конструкторы (по умолчанию и копирования) перестают существовать, а также пропадает стандартная операция присваивания для объектов класса. Поэтому, как только объявляется конструктор перемещения, то дополнительно, как правило, нужно объявлять конструктор по умолчанию, конструктор копирования, операцию присваивания и еще одну операцию присваивания перемещением. Это в программировании называется правилом пяти, которое гласит, что при переопределении одного из следующих пяти методов:

- конструктора копирования;
- деструктор;
- операции присваивания;
- конструктора перемещения;
- операции присваивания перемещением

скорее всего, следует переопределить и четыре остальных.

Давайте посмотрим, как и для чего переопределяется последняя операция присваивания перемещением. Она прописывается для класса DArray следующим образом:

```language-cpp
class DArray {
...
    DArray& operator=(DArray&& right) noexcept
    {
        if (this == &right) return *this;
        std::cout << "DArray move assigment" << std::endl;

        delete[] data;
        length = right.length;
        capacity = right.capacity;
        data = right.data;
        right.data = nullptr;

        return *this;
    }
};
```

И компилятор вызывает ее всякий раз, когда временный объект присваивается текущему объекту. В нашем примере именно так и происходит. При этом сама операция работает по аналогии с конструктором перемещения. Все ресурсы объекта right захватываются текущим объектом без создания копий, что ускоряет процесс присваивания. После этого объект right уже не может использовать свои данные и предполагается, что в программе будет вскоре удален. Поэтому перемещение можно использовать только с временными объектами, которые нет смысла копировать.

Компилятор сам четко различает ситуации, когда какую операцию присваивания вызывать. Например, если ниже в функции main прописать:

```language-cpp
    DArray a;
    a = ard;
```

то здесь будет использована обычная операция присваивания, т.к. слева и справа стоят lvalue выражения.